6.851: Advanced Data Structures, Fall 2017
Prof. Erik Demaine, Adam Hesterberg, Jayson Lynch

Problem Set 2 Solutions

Due: Wednesday, September 20, 2017 at noon

Problem 2.1 [Integer Retroactivity].

Describe and analyze a *fully retroactive* data structure for storing an integer, initially 0. Your data structure should support INSERT(t, *update*) and DELETE(t, *update*), where t denotes the time of the operation and *update* is one of the following update operations:

(a) ADD(x): add x to the stored integer.

(b) MULTIPLY(x): multiply the stored integer by x.

(c) SET(x): set the stored integer to x.

You should be able to query the value of the integer at a time t.

Your data structure should support all operations (retroactive updates and queries) in $O(\log m)$ time per operation, where m denotes the number of update operations (ADD, MULTIPLY, and SET) that have been added to and not deleted from the data structure. The space usage should be $O(m)$.

Solution: Note that all three update operations apply a linear function to the stored integer t:

(a) ADD(x) takes t to $1 \cdot t + x$.

(b) MULTIPLY(x) takes t to $x \cdot t + 0$.

(c) SET(x) takes t to $0 \cdot t + x$.

The composition of such linear functions is linear, so any sequence of updates is equivalent to one linear function, which can be stored in $O(1)$ space and applied in $O(1)$ time.

We store updates in a balanced binary search tree whose keys are the update times, augmented with, for each node, the composition of the functions stored at the nodes in its subtree, which is a linear function as above. This augmentation is a local property of the node (that is, the augmentation at a node depends only on that node itself and the augmentations at its children), so it can be maintained during rotations, insertions, and deletions. To query at time t, walk down the tree to time t, applying the functions of the left children; their composition is the value at time t.